A Genetic Programming Hyper-Heuristic for the Multidimensional Knapsack Problem

John H. Drake, Matthew Hyde, Khaled Ibrahim and Ender Özcan
School of Computer Science
University of Nottingham
Jubilee Campus
Wollaton Road
Nottingham, NG8 1BB, UK
Email: jqd@cs.nott.ac.uk, matthew.hyde@nottingham.ac.uk, kh.hameed@gmail.com, ender.ozcan@nottingham.ac.uk

***Abstract*—Hyper-heuristics are a class of high-level search techniques which operate on a search space of heuristics rather than directly on a search space of solutions. Early hyper-heuristics focussed on selecting and applying a low-level heuristic at each stage of a search. Recent trends in hyper-heuristic research have led to a number of approaches being developed to automatically generate new heuristics from a set of heuristic components. This work investigates the suitability of using genetic programming as a hyper-heuristic methodology to generate constructive heuristics to solve the multidimensional 0-1 knapsack problem. A population of heuristics to rank knapsack items are trained on a subset of test problems and then applied to unseen instances. The results over a set of standard benchmarks show that genetic programming can be used to generate constructive heuristics which yield human-competitive results.**

***Index Terms*—Hyper-heuristics, Genetic Programming, Heuristic Generation, Multidimensional Knapsack Problem**

I. Introduction

Many optimisation problems create a search space which is too large to enumerate and exhaustively search for an optimal solution. A large number of heuristics and meta-heuristics have been successfully applied to such NP-hard problems. A weakness of these approaches is the need to manually adapt the method used in order to solve different problem domains or classes of problem. Hyper-heuristics are an emerging class of high-level search techniques designed to automate the heuristic design process and raise the level of generality at which search methods operate [1]. Hyper-heuristics operate on a search space of heuristics unlike traditional computational search methods which operate directly on a search space of solutions. Hyper-heuristic research is driven by the desire to provide more general search methods which select and create methods for solving problems rather than searching directly for solutions.

Hyper-heuristics can be broadly split into two main categories, those methodologies which select a low-level heuristic to apply at a given point in the search and those methodologies which create new heuristics from a set of low-level components [2]. Here we will be concerned with the latter of these two categories. A motivation for this area of research is that a heuristic can be automatically specialised to a given class of problems, with certain characteristics. It is often prohibitively expensive to manually tune a heuristic methodology for each new class of problem instances, but doing so would produce better results. If we can research methods to successfully automate that design process, then much better results can potentially be obtained, with no extra human effort. Genetic programming is a standard evolutionary computation technique which has been successfully employed to automatically generate heuristics for a number of NP-hard [3] combinatorial optimisation problems [4], [5], [6], [7], [8], [9], [10]. The multidimensional 0-1 knapsack problem is a standard NP-hard problem which is derived from real-world applications such as capital budgeting [11] and project selection [12].

In this paper, we investigate the suitability of genetic programming to evolve reusable constructive heuristics for the multidimensional 0-1 knapsack problem. We compare the performance of the automatically generated heuristics to human-designed constructive heuristics and meta-heuristics from the literature over a set of standard benchmark instances.

Section II provides an overview of hyper-heuristics in general and genetic programming hyper-heuristic in particular. Then the multidimensional 0-1 knapsack problem is described in section III. Section IV provides the experimental design and settings of genetic programming for solving the problem which is followed by section V discussing the results of the computational experiments. Finally, section VI concludes our work.

II. Hyper-heuristics

Denzinger et al. [13] first used the term 'hyper-heuristic' to describe a system which selects and combines a number of artificial intelligence methods. Although this was when the term 'hyper-heuristic' originated, the idea of operating on a search space of heuristics can be traced back to the early 1960's. Fisher and Thompson [14] showed that combining rules for job shop scheduling could yield better results than taking any single individual rule. Cowling et al. [15] introduced the term in the field of combinatorial optimisation and defined hyper-heuristics as '*heuristics to choose heuristics*'. A more recent definition was provided by Burke et al. [2], [16] to include hyper-heuristics which generate new heuristics from components of existing heuristics:

'A hyper-heuristic is a search method or learning mechanism for selecting or generating heuristics to solve computational search problems.'

A. A classification of hyper-heuristic approaches

Burke et al. [2] outline two main categories of hyper-heuristics; heuristic selection methodologies and heuristic generation methodologies. Heuristic selection methodologies select a low-level heuristic to apply at a given point in the search space. Heuristic generation methodologies automatically generate new heuristics from a set of low-level components or building blocks. In either case, the set of low-level heuristics being selected or generated can either be further split to distinguish between those which construct solutions from scratch (constructive) and those which modify an existing solution (perturbative) (see [17] for more). As well as the nature of the search space, hyper-heuristics can learn from feedback concerning heuristic performance throughout the search process. Hyper-heuristics which utilise online learning continuously adapt throughout the search process based on the feedback they receive. Hyper-heuristics using offline learning train a hyper-heuristic on a subset of instances before being applied to a larger set of unseen instances.

B. Genetic programming as a hyper-heuristic

Genetic programming is one of the more recently developed classes of evolutionary algorithms proposed by Koza [18]. Unlike traditional forms of evolutionary computation, populations of computer programs usually expressed as tree structures are evolved. Rather than producing fixed-length encoded representations of candidate solutions to a given problem, the evolved program itself, when executed, is the solution. Burke et al. [19] outline the suitability of genetic programming as a hyper-heuristic to generate new heuristics and survey previous work attempting to create heuristics using genetic programming. One of the advantages highlighted is that genetic programming relies on expert knowledge to define its terminal and function sets. As human expert knowledge is necessary, domain specific information can be incorporated into the fundamental components of the system. A second advantage is that other methods (such as genetic algorithms) may restrict the length of an encoded solution in order to facilitate simple genetic operators, genetic programming trees have variable length representation. This can be useful if the best length encoding for heuristic representation is not known. Finally, genetic programming can be used to evolve trees as executable programs allowing low-level heuristics to be generated directly.

Genetic programming has successfully been used to evolve new constructive heuristics comparable to human designed heuristics for a number of problem domains. Burke et al. [4], [5] showed that stand-alone heuristics generated using genetic programming could outperform the human designed 'best-fit' heuristic from the literature on unseen instances of the same class of one dimensional bin packing problems. This work was extended to three dimensional bin packing by Allen at al. [20] and generalised by Burke et al. [6] to include one, two and three dimensional bin packing problems, again obtaining human competitive results. A similar method was presented by Burke et al. [21] for two dimensional strip packing problems. Bader-El-Din and Poli [8] used genetic programming to quickly generate 'disposable' heuristics to solve the satisfiability problem. Again, this work generated heuristics comparable to those which were human designed. However, only a limited search space of heuristics was covered. Kumar et al. [22] used genetic programming as a hyper-heuristic to evolve heuristics for the biobjective 0-1 knapsack problem. This system successfully created 'reusable' heuristics able to produce a set of Pareto-optimal solutions. The Pareto fronts generated using this approach are indistinguishable from those obtained using the human-desinged profit-to-weight ratio heuristic. Hauptman et al. [10] employ genetic programming to generate solvers for two common puzzles including the NP-Complete Freecell. Genetic programming has also been used as a hyper-heuristic by Keller and Poli [23] for the travelling salesman problem, by Fukunaga [9] to generate local search heuristics for satisfiability and by Geiger et al. [7] to create dispatching rules for the job shop problem. At a higher level of abstraction, Hyde et al. [24] evolve the acceptance criteria component of a selection hyper-heuristic. The evolved acceptance criteria performed well when compared to standard acceptance criteria from the literature on instances of both bin packing and MAX-SAT.

III. The multidimensional 0-1 knapsack problem

The multidimensional 0-1 knapsack problem is an NP-hard [3] combinatorial optimisation problem whereby the objective is to select a subset of items which maximise profit whilst conforming to a number of constraints. Each item consumes a certain amount of resources in each of the knapsacks dimensions and the capacity of each of the knapsacks dimensions must be respected. More formally, a multidimensional 0-1 knapsack problem with n items and m dimensions can be defined as:

$$\text{maximise} \quad \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \qquad i = 1, ..., m \tag{2}$$

$$\text{with} \quad x_j \in \{0, 1\}, \qquad j = 1, ..., n \tag{3}$$

where p_j is the profit for selecting item j, x_1,...,x_n is a set of decision variables indicating whether or not object j is selected, b_i is the capacity of each dimension i and a_{ij} is the resource consumption of item j in dimension i.

Senju and Toyoda [25] proposed a heuristic which starts with all variables x_1,...,x_n set to 1 and successively sets variables to 0 in order of increasing utility value until a feasible solution is found. Magazine and Oguz [26] presented a heuristic algorithm which combined the method of Senju and Toyoda [25] with the Generalised Lagrange Multiplier

approach of Everett [27] to fix certain variables. This work was improved by Volgenant and Zoon [28]. More recently, a variety of exact and meta-heuristic methods have also been proposed in the literature to solve the multidimensional 0-1 knapsack problem including simulated annealing [29], [30], neural networks [31], genetic algorithms [32], memetic algorithms [33], [34], selection hyper-heuristics [35] and particle swarm optimisation [36] and core-based and tree search algorithms [37], [38], [39]. Fréville [40] provides a more complete survey of the multidimensional 0-1 knapsack problem literature.

IV. A GENETIC PROGRAMMING HYPER-HEURISTIC FOR THE MULTIDIMENSIONAL KNAPSACK PROBLEM

A number of papers in the literature [25], [26], [33], [41], [42], [43], [44] make use of an add and (or) $drop$ phase to either construct, improve or repair solutions to the multidimensional 0-1 knapsack problem. These techniques more often than not use a *utility-weight* value to score and sort the objects in order of their relative efficiency. Here we use genetic programming to evolve a population of heuristics which assign a score to each potential knapsack item. Each heuristic is evaluated by attempting to add the items to the knapsack in descending order determined by these scores (highest rank first) over a set of 'training' instances.

A. Genetic programming function and terminal sets

The first four rows of Table I show the function set of the genetic programming runs. The arithmetic operators add, subtract, multiply and protected divide are chosen to be included in the function set. In addition to standard add, subtract and multiply operators we use 'protected divide' instead of the traditional divide function. As there is always a possibility that the denominator could be zero, protected divide replaces zero with 0.001. The rest of Table I shows the terminal set. The $avgDiff$ for item j is calculated as:

$$avgDiff_j = \frac{\sum_{i=1}^{m} b_i - a_{ij}}{m} \quad (4)$$

Depending on the number of dimensions in the set of instances currently being considered there is a set of m $conDim$ terminals for each of $conDim_{j,1}$ ... $conDim_{j,m}$.

B. Experimental design

The ORLib instances introduced by Chu and Beasley [33] are used to test the genetic programming hyper-heuristic. ORLib contains 270 instances with $n \in \{100, 250, 500\}$ variables, $m \in \{5, 10, 30\}$ dimensions and $tightness\ ratio \in \{0.25, 0.50, 0.75\}$. As optimal solutions are not known for these instances the %-gap is used to measure performance. The %-gap is the distance from the upper bound provided by the solution to the related LP-relaxed problem calculated as:

$$100 * \frac{LPopt - SolutionFound}{LPopt} \quad (5)$$

For each set of 10 instances we use 5 to 'train' the hyper-heuristic before applying the best evolved heuristic to the further 5 as yet unseen 'test' instances. The fitness of an individual in the GP population is measured as the sum of the profit obtained on the 5 training instances. The next generation of the genetic programming run is then populated using the best performing heuristics. Table II shows the parameters used in the genetic programming runs. The mutation operator uses the 'grow' method described by Koza [18], with a set depth of five. The crossover operator produces two individuals with a maximum depth of 17.

Each experiment was repeated 5 times for each set of instances. All experiments were carried out on an Intel i7 2 GHz CPU with 6 GB memory using the genetic programming implementation of the ECJ (Evolutionary Computation in Java) package.

TABLE I
FUNCTION AND TERMINAL SETS OF EACH GENETIC PROGRAMMING RUN

Name	Description
+	Add two inputs
-	Subtract second input from first input
*	Multiply two inputs
%	Protected divide function
p_j	Profit of the current item j
$avgDiff_j$	Average difference between the capacity and resource consumption of the current item for each dimension of the knapsack
$conDim_{j,1}$	Resource consumption of the current item j in dimension 1
$conDim_{j,2}$	Resource consumption of the current item j in dimension 2
$conDim_{j,...}$	Resource consumption of the current item j in dimension...
$conDim_{j,m}$	Resource consumption of the current item $j$in dimension m

TABLE II
PARAMETERS OF EACH GENETIC PROGRAMMING RUN

Generations	50
Population Size	10000
Crossover Probability	0.85
Mutation Probability	0.1
Reproduction Probability	0.05
Tree initialisation method	Ramped half-and-half
Selection Method	Tournament Selection, Size 7

V. EXPERIMENTAL RESULTS

Table III shows the average results in terms of %-gap for the best evolved heuristic of 5 runs for each set of instances in ORLib. Each set of instances consists of 5 'training' instances and 5 'test' instances and is labelled as ORmxn with $m \in \{5, 10, 30\}$ dimensions and $n \in \{100, 250, 500\}$ variables. Each mxn combination also varies with $tightness\ ratio \in \{0.25, 0.50, 0.75\}$. From this table we note that a better average percentage gap is obtained when $tightness\ ratio$ increases.

Table IV shows the average %-gap for a number of techniques from the literature over all 270 instances in the ORLib benchmark set. Our approach generates heuristics which can outperform previous human-designed constructive heuristics.

TABLE III
DETAILED PERFORMANCE OF BEST HEURISTICS GENERATED BY GE PROGRAMMING HYPER-HEURISTICS ON ORLIB INSTANCES BASEI AVERAGE %-GAP

Instance Set	*tightness ratio* 0.25	0.50	0.75	Average
OR5x100	4.98	2.05	1.36	2.80
OR5x250	3.08	1.66	0.77	1.84
OR5x500	2.38	1.64	0.71	1.58
OR10x100	7.39	3.54	2.26	4.40
OR10x250	4.43	2.78	1.15	2.79
OR10x500	3.77	1.97	0.99	2.24
OR30x100	8.67	4.70	2.43	5.27
OR30x250	5.73	3.25	1.70	3.56
OR30x500	4.80	2.54	1.40	2.91
All instances	5.03	2.68	1.42	**3.04**

TABLE IV
COMPARISON OF GENETIC PROGRAMMING HYPER-HEURISTIC TO PREVIOUS APPROACHES OVER ALL INSTANCES IN ORLIB IN TERMS OF %-GAP

Type	Reference	%-gap
MIP	Drake et al. [35] (CPLEX 12.2)	0.52
MA	Chu and Beasley [33]	0.54
Selection HH	Drake et al. [35]	0.70
MA	Özcan and Basaran [34]	0.92
Heuristic	Pirkul [41]	1.37
Heuristic	Fréville and Plateau [43]	1.91
Metaheuristic	Qian and Ding [30]	2.28
Generation HH	**Genetic programming hyper-heuristic**	**3.04**
MIP	Chu and Beasley [33] (CPLEX 4.0)	3.14
Heuristic	Akçay et al. [45]	3.46
Heuristic	Volgenant and Zoon [28]	6.98
Heuristic	Magazine and Oguz [26]	7.69

The heuristics generated by our genetic programming hyper-heuristic achieve an average %-gap of 3.04, this is lower than the human-designed constructive heuristic methods proposed by Akçay et al. [45], Volgenant and Zoon [28] and Magazine and Oguz [26]. They can also outperform the primitive MIP applied by Chu and Beasley [33]. Many of the better performing techniques in the literature, such as the MA of Chu and Beasley [33] and the selection hyper-heuristic of Drake et al. [35], make use of a repair operator based on solutions to the LP-relaxed version of the multidimensional 0-1 knapsack problem. Once a reusable constructive heuristic has been evolved it takes considerably less computational effort to find a solution than calculating the LP-relaxed solutions. In this case, the LP-relaxed solutions must be calculated for every new instance encountered. Approaches using LP-relaxed solutions also then require the extra effort of applying a metaheuristic or other technique afterwards. Figure 1 shows the fitness value of the best performing heuristic at each generation of a sample GP run on the OR5x100 set. This plot shows a steady improvement in solution quality with the best-of-run heuristic not found until the latter stages of the run.

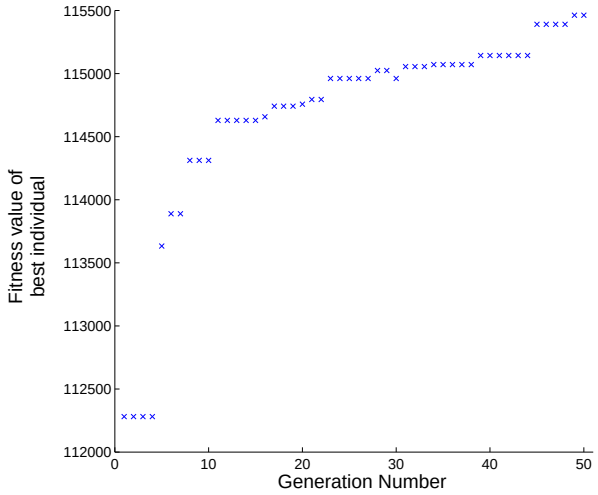

Fig. 1. Best solution found at each generation of a sample run on the on the 5 training instances of the OR5x100 set

VI. CONCLUSIONS

In this work we have shown that genetic programming can be used as a hyper-heuristic to generate reusable constructive heuristics for the multidimensional 0-1 knapsack problem. Our method is classified as a hyper-heuristic approach as it operates on a search space of heuristics rather than a search space of solutions. To the authors knowledge, this is the first time in literature a GP hyper-heuristic has been used to solve the multidimensional 0-1 knapsack problem. This method has shown that automatically generated heuristics can be competitive with human-designed heuristics from the literature. Many methods make use of an add and (or) $drop$ phase to either construct, improve or repair solutions. As future work, the rankings derived from the evolved heuristics can be used to define the order in which items are considered to be added or dropped. Many of the best results in the literature rely on knowledge gained from the LP-relaxed version of the multidimensional 0-1 knapsack problem. We intend to incorporate the optimal LP-relaxed results as part of a more comprehensive function set to attempt to improve the heuristics generated.

REFERENCES

[1] P. Ross, "Hyper-heuristics," in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke and G. Kendall, Eds. Springer, 2005, ch. 17, pp. 529–556.

[2] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. Woodward, *Handbook of Metaheuristics*. Springer, 2010, ch. A Classification of Hyper-heuristics Approaches, pp. 449–468.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[4] E. K. Burke, M. Hyde, and G. Kendall, "Evolving bin packing heuristics with genetic programming," in *Proceedings of PPSN IX*. Springer-Verlag, 2006, pp. 860–869.

[5] E. K. Burke, J. Woodward, M. Hyde, and G. Kendall, "Automatic heuristic generation with genetic programming: Evolving a jack-of-all trades or a master of one," in *GECCO 2007*, 2007, pp. 7–11.

[6] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward, "Automating the packing heuristic design process with genetic programming," *Evolutionary Computation*, vol. 20, no. 1, pp. 63–89, 2011.

[7] C. D. Geiger, R. Uzsoy, and H. Aytug, "Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach," *Journal of Scheduling*, vol. 9, no. 1, pp. 7–34, 2006.

[8] M. Bader-El-Den and R. Poli, "Generating sat local-search heuristics using a gp hyper-heuristic framework," in *Artificial Evolution*, ser. LNCS. Springer Berlin / Heidelberg, 2008, vol. 4926, pp. 37–49.

[9] A. S. Fukunaga, "Automated discovery of local search heuristics for satisfiability testing," *Evolutionary Computation*, vol. 16, no. 1, pp. 31–61, 2008.

[10] A. Hauptman, A. Elyasaf, and M. Sipper, "Evolving hyper heuristic-based solvers for rush hour and freecell," in *Proceedings of SOCS 2010*, 2010, pp. 149–150.

[11] J. Lorie and L. Savage, "Three problems in capital rationing," *Journal of Business*, vol. 28, pp. 229–239, 1955.

[12] C. Petersen, "Computational experience with variants of the balas algorithm applied to the selection of r&d projects," *Management Science*, vol. 13, no. 9, pp. 736–750, 1967.

[13] J. Denzinger, M. Fuchs, and M. Fuchs, "High performance atp systems by combining several ai methods," SEKI-Report SR-96-09, University of Kaiserslautern, Tech. Rep., 1996.

[14] M. Fisher and G. Thompson, "Probabilistic learning combinations of local job-shop scheduling rules," in *Factory Scheduling Conference*, 1961.

[15] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *Proceedings of PATAT 2000*. Springer-Verlag, 2001, pp. 176–190.

[16] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art, to appear in the Journal of the Operational Research Society (2012)."

[17] E. Özcan, B. Bilgin, and E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intelligent Data Analysis*, vol. 12, no. 1, pp. 3–23, 2008.

[18] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA: The MIT Press, 1992.

[19] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. Woodward, *Computational Intelligence: Collaboration, Fusion and Emergence*. Springer-Verlag, 2009, ch. Exploring Hyper-heuristic Methodologies with Genetic Programming, pp. 177–201.

[20] S. Allen, E. K. Burke, M. Hyde, and G. Kendall, "Evolving reusable 3d packing heuristics with genetic programming." in *Proceedings of GECCO 2009*, Montreal, Canada, 2009, pp. 931–938.

[21] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward, "A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 942–958, 2010.

[22] R. Kumar, A. H. Joshi, K. K. Banka, and P. I. Rockett, "Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming," in *Proceedings of GECCO 2008*, 2008, pp. 1227–1234.

[23] R. E. Keller and R. Poli, "Linear genetic programming of parsimonious metaheuristics," in *Proceedings of CEC 2007*, 2007, pp. 4508–4515.

[24] M. Hyde, E. Özcan, and E. K. Burke, "Multilevel search for evolving the acceptance criteria of a hyper-heuristic," in *MISTA 2009*, 2009, pp. 798–801.

[25] S. Senju and Y. Toyoda, "An approach to linear programming with 0-1 variables," *Management Science*, vol. 15, pp. 196–207, 1968.

[26] M. J. Magazine and O. Oguz, "A heuristic algorithm for the multidimensional zero-one knapsack problem," *EJOR*, vol. 16, no. 3, pp. 319–326, 1984.

[27] H. Everett, "Generalized lagrange multiplier method for solving problems of optimum allocation of resources," *Operations Research*, vol. 11, no. 3, pp. 399–417, 1963.

[28] A. Volgenant and J. A. Zoon, "An improved heuristic for multidimensional 0-1 knapsack problems," *Journal of the Operational Research Society*, vol. 41, pp. 963–970, 1990.

[29] A. Drexl, "A simulated annealing approach to the multiconstraint zero-one knapsack problem," *Computing*, vol. 40, no. 1, pp. 1–8, 1988.

[30] F. Qian and R. Ding, "Simulated annealing for the 0/1 multidimensional knapsack problem," *Numerical Mathematics*, vol. 16, pp. 320–327, 2007.

[31] M. Ohlsson, C. Peterson, and B. Söderberg, "Neural networks for optimization problems with inequality constraints: the knapsack problem," *Neural Comput.*, vol. 5, pp. 331–339, 1993.

[32] S. Khuri, T. Bäck, and J. Heitkötter, "The 0/1 multiple knapsack problem and genetic algorithms," in *Proceedings of SAC 1994*. New York, USA: ACM, 1994, pp. 188–193.

[33] P. C. Chu and J. E. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *Journal of Heuristics*, vol. 4, no. 1, pp. 63–86, 1998.

[34] E. Özcan and C. Başaran, "A case study of memetic algorithms for constraint optimization," *Soft Computing*, vol. 13, no. 8-9, pp. 871–882, 2009.

[35] J. H. Drake, E. Özcan, and E. K. Burke, "Controlling crossover in a selection hyper-heuristic framework," School of Computer Science, University of Nottingham, Tech. Rep. No. NOTTCS-TR-SUB-1104181638-4244, 2011.

[36] F. Hembecker, H. S. Lopes, and W. Godoy, Jr., "Particle swarm optimization for the multidimensional knapsack problem," in *Proceedings of ICANNGA 2007*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 358–365.

[37] Y. Vimont, S. Boussier, and M. Vasquez, "Reduced costs propagation in an efficient implicit enumeration for the 0-1 multidimensional knapsack problem," *J. of Combinatorial Opt.*, vol. 15, pp. 165–178, 2008.

[38] S. Boussier, M. Vasquez, Y. Vimont, S. Hanafi, and P. Michelon, "A multi-level search strategy for the 0-1 multidimensional knapsack problem," *Discrete Applied Mathematics*, vol. 158, no. 2, pp. 97–109, 2010.

[39] R. Mansini and M. G. Speranza, "Coral: An exact algorithm for the multidimensional knapsack problem," *INFORMS J. on Computing*, 2011, to Appear.

[40] A. Fréville, "The multidimensional 0-1 knapsack problem: An overview," *EJOR*, vol. 155, no. 1, pp. 1–21, 2004.

[41] H. Pirkul, "A heuristic solution procedure for the multiconstraint zero-one knapsack problem," *Naval Research Logistics*, vol. 34, no. 2, pp. 161–172, 1987.

[42] S. Hanafi and A. Freville, "An efficient tabu search approach for the 0-1 multidimensional knapsack problem," *EJOR*, vol. 106, no. 2-3, pp. 659–675, 1998.

[43] A. Fréville and G. Plateau, "An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem," *Discrete Applied Mathematics*, vol. 49, no. 1-3, pp. 189–212, 1994.

[44] F. Glover and G. Kochenberger, "Critical event tabu search for multidimensional knapsack problems," in *Meta-Heuristics: Theory & Applications*. Kluwer Academic Publishers, 1996, pp. 407–427.

[45] Y. Akçay, H. Li, and S. H. Xu, "Greedy algorithm for the general multidimensional knapsack problem," *Annals of Operations Research*, vol. 150, no. 1, pp. 17–29, 2007.